

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术201707**

**学 号： U201714785**

**姓 名： 聂鸿勇**

**指导教师： 祝建华**

**报告日期： 2018.11.20**

**计算机科学与技术学院**

目 录

[1 基于顺序存储结构的线性表实现 2](#_Toc458159879)

[1.1 问题描述 2](#_Toc458159880)

[1.2 系统设计 2](#_Toc458159882)

[1.3 系统实现 2](#_Toc458159883)

[1.4 实验小结 2](#_Toc458159884)

[2 基于链式存储结构的线性表实现 2](#_Toc458159885)

[2.1 问题描述 2](#_Toc458159886)

[2.2 系统设计 2](#_Toc458159887)

[2.3 系统实现 2](#_Toc458159888)

[2.4 实验小结 2](#_Toc458159889)

[3 基于二叉链表的二叉树实现 2](#_Toc458159890)

[3.1 问题描述 2](#_Toc458159891)

[3.2 系统设计 2](#_Toc458159892)

[3.3 系统实现 2](#_Toc458159893)

[3.4 实验小结 2](#_Toc458159894)

[4 基于二叉链表的二叉树实现 2](#_Toc458159895)

[4.1 问题描述 2](#_Toc458159896)

[4.2 系统设计 2](#_Toc458159897)

[4.3 系统实现 2](#_Toc458159898)

[4.4 实验小结 2](#_Toc458159899)

[参考文献 2](#_Toc458159900)

[附录A 基于顺序存储结构线性表实现的源程序 2](#_Toc458159901)

[附录B 基于链式存储结构线性表实现的源程序 2](#_Toc458159902)

[附录C 基于二叉链表二叉树实现的源程序 2](#_Toc458159903)

[附录D 基于邻接表图实现的源程序 2](#_Toc458159904)

# 1 基于顺序存储结构的线性表实现

# 1.1 问题描述

线性表在物理内存中可以以顺序表的方式实现，即物理上存储位置相邻的两个元素是线性表中的相邻元素，且数据元素的前后关系不变。

## 1.1.1实验要完成的顺序表算法：

⑴初始化表：函数名称是InitaList(L)；初始条件是线性表L不存在已存在；操作结果是构造一个空的线性表。

⑵销毁表：函数名称是DestroyList(L)；初始条件是线性表L已存在；操作结果是销毁线性表L。

⑶清空表：函数名称是ClearList(L)；初始条件是线性表L已存在；操作结果是将L重置为空表。

⑷判定空表：函数名称是ListEmpty(L)；初始条件是线性表L已存在；操作结果是若L为空表则返回TRUE,否则返回FALSE。

⑸求表长：函数名称是ListLength(L)；初始条件是线性表已存在；操作结果是返回L中数据元素的个数。

⑹获得元素：函数名称是GetElem(L,i,e)；初始条件是线性表已存在，1≤i≤ListLength(L)；操作结果是用e返回L中第i个数据元素的值。

⑺查找元素：函数名称是LocateElem(L,e,compare())；初始条件是线性表已存在；操作结果是返回L中第1个与e满足关系compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为0。

⑻获得前驱：函数名称是PriorElem(L,cur\_e,pre\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre\_e无定义。

⑼获得后继：函数名称是NextElem(L,cur\_e,next\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它的后继，否则操作失败，next\_e无定义。

⑽插入元素：函数名称是ListInsert(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

⑿遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。

## 1.1.2实验目标：

通过实验达到⑴加深对线性表的概念、基本运算的理解；⑵熟练掌握线性表的逻辑结构与物理结构的关系；⑶物理结构采用顺序表,熟练掌握线性表的基本运算的实现。

# 1.2 系统设计

## 1.2.1总体架构

本系统采用顺序表作为线性表的物理结构，实现线性表的基本运算。

系统具有一个功能菜单。在主程序中完成函数调用所需实参值的准备和函数执行结果的现实，并给出适当的操作提示显示。

系统通过定义一个SqList类型的含有线性表指针数组和当前线性表数量的结构体。

系统开始运行时调用函数读取文件中的数据，并提供数据保存功能以实现线性表的文件形式保存。

该演示系统提供的操作有：表的初始化、销毁、清空、判空，求表长、获取数据元素、查找数据元素、获得前驱、获得后继、插入数据元素、删除数据元素、表的遍历、表的选择、数据保存。

在程序中实现消息处理和操作提示，包括数据的输入和输出，错误操作提示、程序的退出。

## 1.2.2数据结构设计

根据实验的要求以及线性表ADT的定义，给出基于C语言实现的线性表ADT的结构定义：

**typedef struct**{  
 ElemType \* elem;  
 **int** length;  
 **int** listSize;  
 **int** IsInitial;  
 }SqList;

其中，Element的定义由具体的需求决定，在本次实验的中，Element的定义如下：

**typedef struct**{  
 **char** number[10]; //学号  
 **char** name[10]; //姓名  
 }ElemType;

实现简单多表操作的TableList的定义如下：

**typedef struct**{  
 SqList \* table;  
 **int** tableSize;  
 **int** length;  
 **int** MaxSize;  
 }TableList;

## 1.2.3 ADT操作实现

InitList：

函数接收一个SqList指针，以传入的指针初始化一个线性表。

通过动态分配空间的方式分配一个Element数组，并且将首地址存储于该线性表结构的成员变量elem中，判断分配空间是否成功，否则报错，最后配置该线性表的长度和初始大小，并给标识量IsInitial赋值为1（表示该线性表已被初始化）。

该线性表的初始长度为0，初始大小由宏LIST\_INIT\_SIZE给定。

DestroyList：

函数接收一个SqList指针，以传入的指针销毁一个线性表。

由主程序条件判断是否已被初始化，如已初始化则进行该操作，释放动态分配空间的数组，并将标识量IsInitial赋值为0（表示该线性表已被销毁）。

ClearList：

函数接收一个SqList指针，以传入的指针清空一个线性表。

首先判断线性表是否存在（IsInitial ==1），若线性表非空，则将长度length=0，可以看作是把数组中的元素清空。

ListEmpty：

该操作接收一个SqList结构变量，判断该线性表是否为空，并且将判断结果存入Bool指针所指向的变量中。

首先判断线性表是否存在（IsInitial ==1），如果存在则判断（length==0 ？）如果为真，返回OK，输出该线性表不为空，反之返回FALSE，输出该线性表为空。

ListLength：

函数接收一个SqList结构变量和一个int指针，计算该线性表的长度，并且将长度值存入int指针所指向的变量中。

首先判断线性表是否存在（IsInitial ==1），如果存在将length值赋给该int指针所指变量中。

GetElem：

该操作接收一个SqList结构变量、一个int变量和一个ElemType指针，获取线性表中特定下标的元素，并且将该元素存入Element指针所指向的变量中。

首先判断线性表是否存在（IsInitial ==1），如果存在判断length是否为0或int值<1或i>length，如果成立则输出所需信息不存在，反之将线性表中对应下标的元素存入ElemType指针所指向的变量中。

LocateElem：

该操作接收一个SqList结构变量、一个int指针、一个ElemType结构变量。

首先判断线性表是否存在（IsInitial ==1），如果存在遍历线性表并调用compare函数，如果找到符合条件的第一个元素则将对应下标赋给int指针指向的对象，反之，报错，输出提示语句。

PriorElem：

该操作接收一个SqList结构变量、一个ElemType结构变量、一个ElemType指针。

首先判断线性表是否存在（IsInitial ==1），如果存在遍历整个线性表，判断线性表中的每个元素是否与传入的ElemType变量满足关系Compare，如果满足则直接结束遍历，判断该元素是否为线性表第一个元素，如果是则输出该元素无前驱，反之，并且将该元素的前驱存入ElemType指针所指的变量中。若没找到符合条件的元素，则输出该信息不存在于线性表中。

NextElem：

该操作接收一个SqList结构变量、一个ElemType结构变量、一个ElemType指针。

首先判断线性表是否存在（IsInitial ==1），如果存在遍历整个线性表，判断线性表中的每个元素是否与传入的ElemType变量满足关系Compare，如果满足则直接结束遍历，判断该元素是否为线性表最后一个元素，如果是则输出该元素无后继，反之，并且将该元素的后继存入ElemType指针所指的变量中。若没找到符合条件的元素，则输出该信息不存在于线性表中。

ListInsert：

该操作接收一个SqList结构变量、一个int变量和一个ElemType结构变量。

首先判断线性表是否存在（IsInitial ==1），如果存在判断输入的int值是否符合条件，若符合，则从后向前将每个元素向后移一个单元，直到将int对应的元素向后移动停止，然后将该ElemType结构变量插入int对应的位置。

ListDelete：

该操作接收一个SqList结构变量、一个int变量和一个ElemType指针。

首先判断线性表是否存在（IsInitial ==1），如果存在判断输入的int值是否符合条件，若符合，则将下标为int的元素赋给ElemType指针所指向的结构变量，然后遍历将该元素后面所有元素向前移一个单元。

ListTraverse：

该操作接收一个SqList结构变量。

首先判断线性表是否存在（IsInitial ==1），如果存在遍历线性表依次对线性表中的每个元素调用Visit函数。  
compare:

该操作接收两个ElemType结构变量，利用strcmp函数判断这两个结构变量是否相等，若是返回OK,反之，返回FALSE。

Visit：

该操作接收一个ElemType结构变量，输出打印出该变量信息。  
Multitable：

该操作接收一个TableList指针、一个int变量。

令MaxSize以及length等于int变量的值，释放该TableList指针原来指向的对象，然后重新分配空间大小为T->MaxSize\***sizeof**(SqList)。并将tableSize赋值为0，完成选择多表操作后，默认表为表一。  
ChooseTable：

该操作接收一个TableList指针、一个int变量。

判断输入int对象的值是否合法，如果合法，则令tableSize的值等于int对象的值。完成对表操作的切换。

## 1.2.4文件存储与读取函数

fwrite(L.elem, **sizeof**(ElemType), (size\_t) L.length, fp);

fread(&L->elem[L->length], **sizeof**(ElemType), 1, fp)；

## 1.2.5源代码文件分类

头文件：LinearTable.h

函数实现：LinearTable.c

主函数：main.c

# 1.3 系统实现

## 1.3.1源代码

**LinearTable.h**:

#ifndef CLION\_FILE\_LINEARTABLE\_H  
#define CLION\_FILE\_LINEARTABLE\_H  
  
#define TRUE 1  
#define FALSE 0  
#define OK 1  
#define ERROR -1  
#define INFEASTABLE -1  
#define OVERFLOW -2  
  
**typedef int** status;  
**typedef struct**{  
 **char** number[10];  
 **char** name[10];  
}ElemType;  
  
#define LIST\_INIT\_SIZE 100  
#define LISTINCREMENT 10  
**typedef struct**{  
 ElemType \* elem;  
 **int** length;  
 **int** listSize;  
 **int** IsInitial;  
}SqList;  
  
**typedef struct**{  
 SqList \* table;  
 **int** tableSize;  
 **int** length;  
 **int** MaxSize;  
}TableList;  
  
status IntiaList(SqList \*L);  
status DestroyList(SqList \*L);  
status ClearList(SqList \*L);  
status ListEmpty(SqList L);  
status ListLength(SqList L);  
status GetElem(SqList L,**int** i,ElemType \*e);  
status LocateElem(SqList L,ElemType e);  
status PriorElem(SqList L,ElemType cur,ElemType \*pre\_e);  
status NextElem(SqList L,ElemType cur,ElemType \*next\_e);  
status ListInsert(SqList \*L,**int** i,ElemType e);  
status ListDelete(SqList \*L,**int** i,ElemType \*e);  
status ListTrabverse(SqList L);  
status Readin(SqList \*L);  
status Writein(SqList L);  
status compare(ElemType L, ElemType m);

status visit(ElemType L);  
status Multitable(TableList \*T, **int** n);  
status ChooseTable(TableList \*T, **int** n);  
  
#endif //CLION\_FILE\_LINEARTABLE\_H

**LinearTable.c:**

#include <malloc.h>  
#include <stdio.h>  
#include <mem.h>  
#include "LinearTable.h"  
  
status IntiaList(SqList \*L){  
 L->elem=(ElemType\*)malloc(LIST\_INIT\_SIZE\***sizeof**(ElemType));  
 **if**( !L->elem ) exit(OVERFLOW);  
 L->listSize = LIST\_INIT\_SIZE;  
 L->length = 0;  
 printf("Enter the information! (EOF end): \n");  
 **while**(scanf("%s %s", L->elem[L->length].number, L->elem[L->length].name)!=EOF){  
 **if**(L->length< LIST\_INIT\_SIZE-1){  
 printf("successfully!\n");  
 L->length++;  
 printf("Enter the information! (EOF end): \n");  
 }  
 **else**{  
 printf("Is full!\n");  
 **break**;  
 }  
 }  
 L->IsInitial = 1;  
 **return** OK;  
}  
  
status DestroyList(SqList \*L){  
 free(L->elem);  
 L->IsInitial = 0;  
 **return** OK;  
}  
  
status ClearList(SqList \*L){  
 L->length = 0;  
 **return** OK;  
}  
  
status ListEmpty(SqList L){  
 **if** (L.length == 0 ) **return** TRUE;  
 **else return** FALSE;  
}  
  
status ListLength(SqList L){  
 **return** L.length;  
}  
  
status GetElem(SqList L,**int** i, ElemType \*e){  
 **if**(L.length == 0 || i < 1 || L.length < i){  
 printf("This information does not exist!\n");  
 **return** ERROR;  
 }  
 \*e = L.elem[i-1];  
 **return** OK;  
}  
  
status LocateElem(SqList L, ElemType e){  
 **for**(**int** i = 1; i <= L.length; i++){  
 **if**(compare(L.elem[i-1], e) == OK){  
 **return** i;  
 }  
 }  
 **return** ERROR;  
}  
  
status PriorElem(SqList L, ElemType cur, ElemType \*pre\_e){  
 **if** (compare(L.elem[0], cur)) {  
 printf("Don't have PriorElem!\n");  
 **return** FALSE;  
 }  
 **for**(**int** i = 0; i < L.listSize; i++){  
 **if**(compare(L.elem[i], cur)){  
 \*pre\_e = L.elem[i - 1];  
 **return** OK;  
 }  
 }  
 **return** ERROR;  
}  
  
status NextElem(SqList L, ElemType cur, ElemType \*next\_e){  
 **for**(**int** m = 0; m < L.length ; m++){  
 **if**(compare(L.elem[m], cur) == OK){  
 **if**(m == L.length - 1){  
 printf("Don't have NextElem!\n");  
 **return** FALSE;  
 }  
 \*next\_e = L.elem[m +1];  
 **return** OK;  
 }  
 }  
 **return** ERROR;  
}  
  
status ListInsert(SqList \*L,**int** i, ElemType e){  
 **if** (!L->elem) **return** ERROR;  
 **else if** (i < 1 || i > L->length + 1) {  
 printf("Enter a valid number!\n");  
 **return** ERROR;  
 }  
 **else** {  
 **if**(L->length >= L->listSize) {  
 ElemType \*newbase;  
 newbase = (ElemType\*)realloc(L->elem,(L->listSize + LISTINCREMENT)\* **sizeof**(ElemType));  
 **if**(!newbase) exit(OVERFLOW);  
 L->elem = newbase;  
 L->listSize += LISTINCREMENT;  
 }  
 **for**(**int** m = 0; m <= L->length - i ; m++){  
 L->elem[L->length-m] = L->elem[L->length-m-1];  
 }  
 }  
 L->elem[i-1] = e;  
 ++L->length;  
 **return** OK;  
}  
  
status ListDelete(SqList \*L, **int** i, ElemType \*e){  
 **if** (!L) **return** ERROR;  
 **else if**(L->length == 0){  
 **return** ERROR;  
 }  
 **else if**(i > L->length || i < 1) **return** ERROR;  
 \*e = L->elem[i-1];  
 **for**(**int** j = i-1; j < L->length-1; j++)  
 L->elem[j] = L->elem[j-1];  
 L->length--;  
 **return** OK;  
}  
  
status ListTrabverse(SqList L){  
 **int** i;  
 printf("\n----------- all elements -----------------------\n");  
 **for** (i = 0; i < L.length; i++) visit(L.elem[i]);

printf("\n------------------ end ------------------------\n");  
 **return** L.length;  
}  
  
status Writein(SqList L){  
 FILE \*fp; **char** filename[30];  
 printf("input file name: ");  
 scanf("%s",filename);  
 **if** ((fp=fopen(filename,"wb"))==NULL)  
 {  
 printf("File open erroe\n ");  
 **return** ERROR;  
 }  
 fwrite(L.elem, **sizeof**(ElemType), (size\_t) L.length, fp);  
 fclose(fp);  
 **return** OK;  
}  
  
status Readin(SqList \*L){  
 FILE \*fp;  
 L->length=0;**char** filename[30];  
 printf("input file name: ");  
 scanf("%s",filename);  
 **if** ((fp=fopen(filename,"rb"))==NULL)  
 {  
 printf("File open erroe\n ");  
 **return** ERROR;  
 }  
 **while**(fread(&L->elem[L->length],**sizeof**(ElemType),1,fp))  
 L->length++;  
//这里从文件中逐个读取数据元素恢复顺序表，对于不同的物理结构，可通过读//取的数据元素恢复内存中的物理结构。  
 fclose(fp);  
 **return** OK;  
}  
  
status compare(ElemType L, ElemType m ){  
 **if**((strcmp(L.number, m.number) == 0) && (strcmp(L.name, m.name) == 0)) **return** OK;  
 **else return** FALSE;  
}

status visit(ElemType L){  
 printf("%s %s\n", L.name, L.number);  
 **return** OK;  
}

status Multitable(TableList \*T, **int** n){  
 T->MaxSize = n;  
 free(T);  
 T->table=(SqList\*)malloc(T->MaxSize\***sizeof**(SqList));  
 T->length = T->MaxSize;  
 T->tableSize = 0;  
}  
  
status ChooseTable(TableList \*T, **int** n){  
 **if**(n > T->length || n < 1) {  
 printf("Enter the right number(0-%d)\n", T->length);  
 **return** ERROR;  
 }  
 T->tableSize = n - 1;  
 **return** OK;  
}

**Main.c:**

#include <stdio.h>  
#include <stdlib.h>  
#include <malloc.h>  
#include <mem.h>  
#include "LinearTable.h"  
  
int main() {  
 TableList T;  
 T.MaxSize = LISTINCREMENT;  
 T.table=(SqList\*)malloc(sizeof(SqList));  
 T.length = 1;  
 T.tableSize = 0;  
 int op = 1;  
 while (op) {  
 system("cls"); printf("\n\n");  
 printf(" Menu for Linear Table On Sequence Structure \n");  
 printf("-------------------------------------------------\n");  
 printf(" 1. IntiaList 7. LocateElem\n");  
 printf(" 2. DestroyList 8. PriorElem\n");  
 printf(" 3. ClearList 9. NextElem \n");  
 printf(" 4. ListEmpty 10. ListInsert\n");  
 printf(" 5. ListLength 11. ListDelete\n");  
 printf(" 6. GetElem 12. ListTrabverse\n");  
 printf(" 13.SaveData 14. ReadData\n");  
 printf(" 15.ChooseTable 16.Multitable\n");  
 printf(" 0. Exit\n");  
 printf("-------------------------------------------------\n");  
 printf("Please choose your number: [0~15]:");  
 scanf("%d", &op);  
 switch(op){  
 case 1:  
 if(T.table[T.tableSize].IsInitial != 1) {  
 if (IntiaList(&T.table[T.tableSize]) == OK) {  
 printf("The linear table was created successfully!\n");  
 } else printf("Linear table creation failed!\n");  
 }  
 else{  
 printf("Linear table already exists!\n");  
 }  
 system("pause");  
 break;  
 case 2:  
 if(T.table[T.tableSize].IsInitial == 1){  
 if(DestroyList(&T.table[T.tableSize]) == OK){  
 printf("Linear table deletion successfully!\n");  
 }  
 else printf("Linear table deletion failed!\n");  
 }  
 else{  
 printf("Linear table does not exist!\n");  
 }  
 system("pause");  
 break;  
 case 3:  
 if(T.table[T.tableSize].IsInitial == 1){  
 if(ClearList(&T.table[T.tableSize]) == OK) printf("Linear table emptying successfully!\n");  
 else printf("Linear table emptying failed!\n");  
 }  
 else{  
 printf("Linear table does not exist!\n");  
 }  
 system("pause");  
 break;  
 case 4:  
 if(T.table[T.tableSize].IsInitial == 1){  
 if(ListEmpty(T.table[T.tableSize]) == OK) printf("Linear table is an empty table\n");  
 else printf("Linear table is not an empty table\n");  
 }  
 else{  
 printf("Linear table does not exist");  
 }  
 system("pause");  
 break;  
 case 5:  
 if(T.table[T.tableSize].IsInitial == 1){  
 printf("Linear-table's length is: %d\n", ListLength(T.table[T.tableSize]));  
 }  
 else{  
 printf("Linear table does not exist!\n");  
 }  
 system("pause");  
 break;  
 case 6:  
 if(T.table[T.tableSize].IsInitial == 1) {  
 int m;  
 ElemType x;  
 printf("Where you want to get the number: \n");  
 scanf("%d",&m);  
 if(GetElem(T.table[T.tableSize], m, &x) == OK) {  
 printf("The number is: %s %s\n", x.number, x.name);  
 } else{  
 printf("The number is not exist!\n");  
 }  
 }  
 else{  
 printf("Linear table does not exist!\n");  
 }  
 system("pause");  
 break;  
 case 7:  
 if(T.table[T.tableSize].IsInitial == 1) {  
 //printf("\n----LocateElem功能待实现！\n");  
 printf("Enter the number you want: ");  
 ElemType e;  
 scanf("%s %s", e.number, e.name);  
 int g;  
 g = LocateElem(T.table[T.tableSize], e);  
 if(g == ERROR)  
 printf("The number is not exit!\n");  
 else{  
 printf("%d\n", g);  
 }  
 }  
 else{  
 printf("Linear table does not exist!\n");  
 }  
 system("pause");  
 break;  
 case 8:  
 if(T.table[T.tableSize].IsInitial == 1){  
 //printf("\n----PriorElem功能待实现！\n");  
 printf("Enter the information: ");  
 ElemType m, n;  
 int h;  
 scanf("%s %s", n.number, n.name);  
 h = PriorElem(T.table[T.tableSize], n, &m);  
 if(h == OK){  
 printf("%s %s\n", m.number, m.name);  
 }  
 else if(h == ERROR)  
 printf("This information does not exist!\n");  
 }  
 else{  
 printf("Linear table does not exist!\n");  
 }  
 system("pause");  
 break;  
 case 9:  
 if(T.table[T.tableSize].IsInitial == 1){  
 // printf("\n----NextElem功能待实现！\n");  
 printf("Enter the information: ");  
 ElemType m,n;  
 int h;  
 scanf("%s %s", n.number, n.name);  
 h = NextElem(T.table[T.tableSize], n, &m);  
 if(h == OK){  
 printf("%s %s\n", m.number, m.name);  
 }  
 else if(h == ERROR)  
 printf("This information does not exist!\n");  
 }  
 else{  
 printf("Linear table does not exist!\n");  
 }  
 system("pause");  
 break;  
 case 10:  
 if(T.table[T.tableSize].IsInitial == 1){  
 printf("Enter the location and number to insert(Location and information): ");  
 int n;  
 scanf("%d", &n);  
 ElemType m;  
 scanf("%s%s", m.number, m.name);  
 if(ListInsert(&T.table[T.tableSize], n, m) == OK)  
 printf("Added successfully!\n");  
 }  
 else{  
 printf("Linear table does not exist!\n");  
 }  
 system("pause");  
 break;  
 case 11:  
 if(T.table[T.tableSize].IsInitial == 1) {  
 if (T.table[T.tableSize].length == 0) {  
 printf("Empty Table!\n");  
 } else {  
 printf("The location you want to delete: \n");  
 int m;  
 ElemType n;  
 scanf("%d", &m);  
 if (ListDelete(&T.table[T.tableSize], m, &n) == OK)  
 printf("%s %s successfully deleted!\n", n.number, n.name);  
 }  
 }  
 else{  
 printf("Linear table does not exist!\n");  
 }  
 system("pause");  
 break;  
 case 12:  
 if(T.table[T.tableSize].IsInitial == 1){  
 if(!ListTrabverse(T.table[T.tableSize])) printf("Linear table is an empty table!\n");  
 }  
 else{  
 printf("Linear table does not exist!\n");  
 }  
 system("pause");  
 break;  
 case 13:  
 if(T.table[T.tableSize].IsInitial == 1){  
 if(Writein(T.table[T.tableSize]) == OK) printf("The data saved successfully!\n");  
 }  
 else{  
 printf("ERROR\n");  
 }  
 system("pause");  
 break;  
 case 14:  
 if(T.table[T.tableSize].IsInitial == 1){  
 if(Readin(&T.table[T.tableSize]) == OK) printf("The data read successfully!\n");  
 }  
 else{  
 printf("ERROR\n");  
 }  
 system("pause");  
 break;  
 case 15:  
 printf("Please enter the table number: \n");  
 int n;  
 scanf("%d", &n);  
 if (n == T.tableSize + 1) {  
 printf("You already in this Linear Table!\n");  
 } else if (ChooseTable(&T, n) == OK)  
 printf("change successful!\n");  
 else {  
 printf("ERROR\n");  
 }  
 system("pause");  
 break;  
 case 16:  
 printf("Number of tables operated at the same time: ");  
 int m;  
 scanf("%d", &m);  
 if(Multitable(&T, m) == OK){  
 printf("successfully!\n");  
 }  
 system("pause");  
 break;  
 case 0:  
 break;  
 default:  
 printf("Please select the right action number you need!\n");  
 system("pause");  
 break;  
 }  
 }  
 printf("Welcome to use this system next time!\n");  
 system("pause");  
 return 0;  
}

## 1.3.2测试数据

演示系统的界面如下图所示：

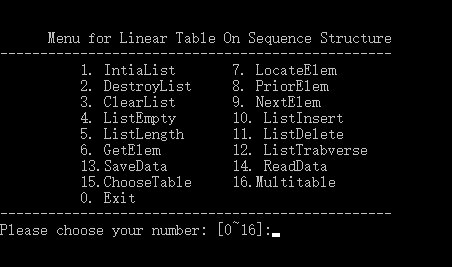


表1：

1,a 2,s 3,d 4,f 5,g 3,d

表2

1,a 2,s 3,d 4.f 5.g

表3

8,a 8,s 8,d 8,f 8,g 8,h

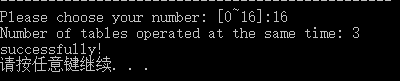
测试用例及其结果如下（各函数测试为独立测试，测试初始数据相同，不受上个函数测试影响）：

## 1.3.3测试操作

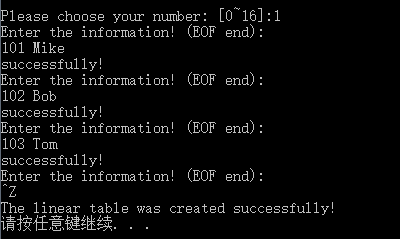
1. 选择多表操作，同时输入需要表的个数3.
2. 初始化默认线性表一，在这项操作中可以输入数据直接给线性表对象赋值（当输入EOF结束）。
3. 遍历线性表。
4. 将线性表清除。
5. 判断线性表是否为空以及测试线性表的长度。
6. 依次将测试数据插入线性表下标为1的位置。
7. 再判断线性表是否为空以及测试线性表的长度。
8. 遍历线性表。
9. 得到下标为4的元素 以及 得到下标超出边界的元素。
10. 获取第一个等于3，d的元素的下标。
11. 获取第一个等于4，f的元素的前驱。
12. 获取第一个等于5，g的元素的后继。
13. 获取第一个等于3，d的元素的前驱。
14. 获取第一个等于1，a的元素的后继。
15. 获取没有存在的元素信息的前驱。
16. 获取没有存在的元素信息的后继。
17. 删除下标为1和5的元素。
18. 遍历线性表。
19. 将线性表存入文件。
20. 遍历线性表。
21. 清空线性表。
22. 对空表进行删除操作
23. 从文件中加载线性表。
24. 判断线性表是否为空以及测试线性表的长度。
25. 遍历线性表。
26. 选择对表二进行操作。
27. 初始化表二并输入测试数据。
28. 选择对表三进行操作。
29. 初始化表三并输入测试数据。
30. 重新返回第二张表并遍历。
31. 重新返回第一张表并遍历。
32. 选择向第五张不存在的表转换。
33. 销毁第一张表。
34. 对第一张表进行各种操作（除初始化）。

## 1.3.2测试结果

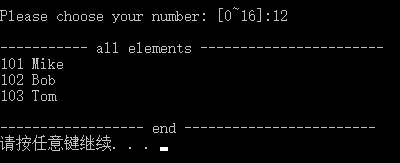
一、选择多表操作成功并选择可以对三个表操作：（选择多表操作成功）



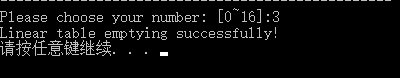
二、初始化并输入第一个表的操作数据（类比于：学号、姓名）：（输入成功）



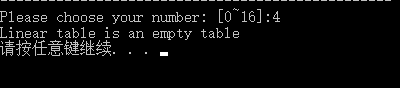
1. 遍历线性表：（遍历成功）

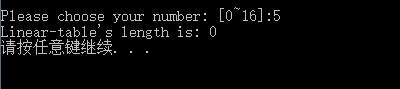


四、将线性表清除：（清楚成功）

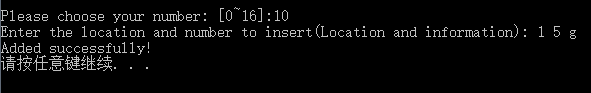


五、判断线性表是否为空以及测试线性表的长度。（空表，长度为0）

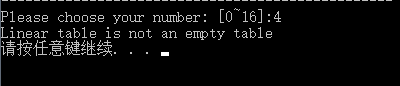


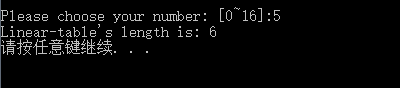


六、依次将测试数据插入线性表下标为1的位置（只显示最后一个元素的插入）：（插入成功）

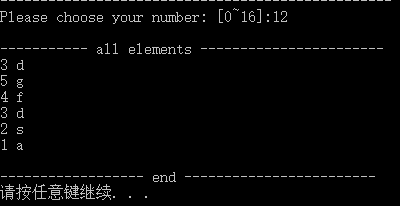


七、再判断线性表是否为空以及测试线性表的长度：（不为空表，长度为6）

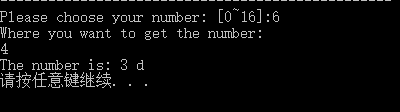




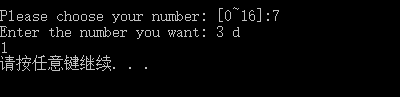
八、遍历线性表：（遍历成功）



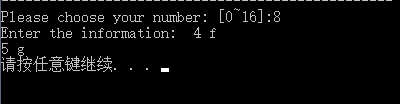
九、得到下标为4的元素：（元素值 3 d）



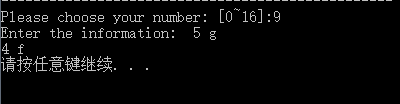
十、获取第一个等于3，d的元素的下标：（下标 1）



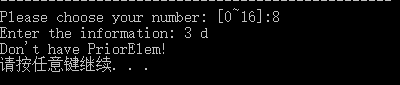
十一、获取第一个等于4，f的元素的前驱：（前驱 5 f）



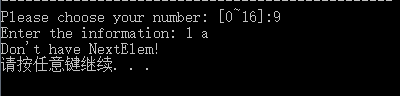
十二、获取第一个等于5，g的元素的后继：（后继 4 f）



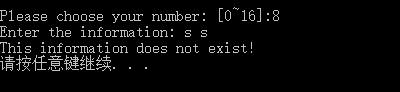
十三、获取第一个等于3，d的元素的前驱：（第一个元素无前驱）



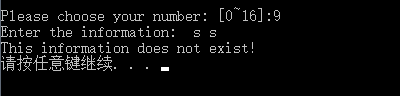
十四、获取第一个等于1，a的元素的后继：（最后一个元素无后继）



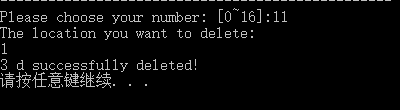
十五、获取没有存在的元素信息的前驱：（信息不存在）

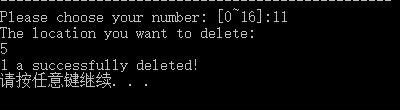


十六、获取没有存在的元素信息的后继：（信息不存在）

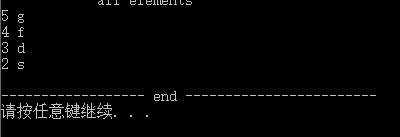


十七、删除下标为1和5的元素：（删除成功）

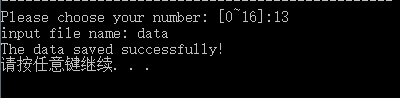




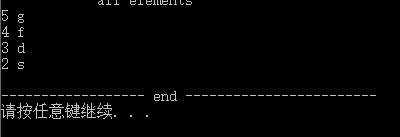
十八、遍历线性表：（遍历成功）



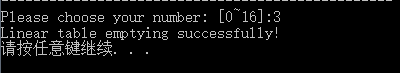
十九、将线性表存入文件data：（保存成功）



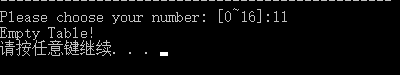
二十、遍历线性表：（遍历成功）



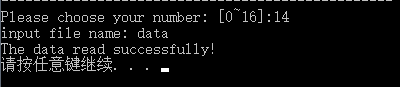
二十一、清空线性表：（清空成功）



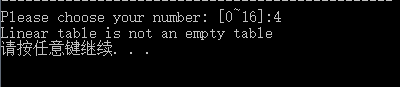
二十二、对空表进行删除操作：（操作失败，提示该表已为空表）

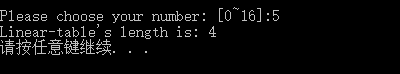


二十三、从文件中加载线性表：（读取成功）

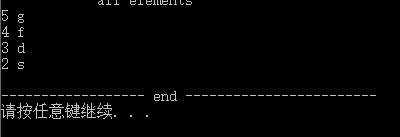


二十四、判断线性表是否为空以及测试线性表的长度：（不为空表，长度为4）

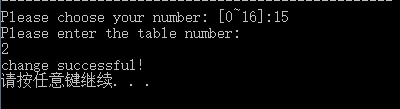




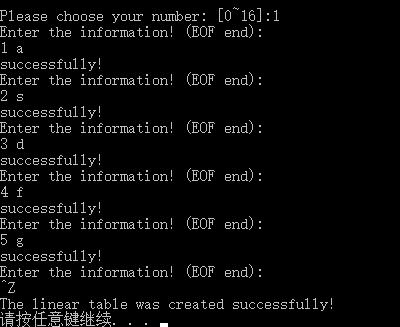
二十五、遍历线性表：（遍历成功）



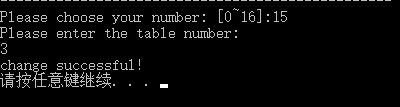
二十六、选择第二个表进行操作：（选择成功）



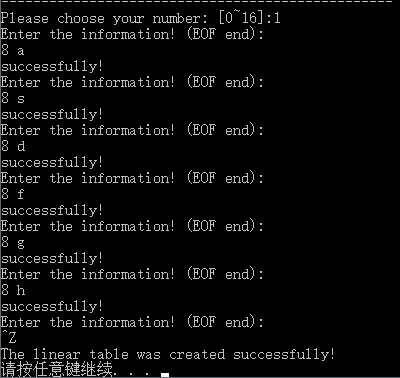
二十七、对第二个表初始化：（初始化成功）



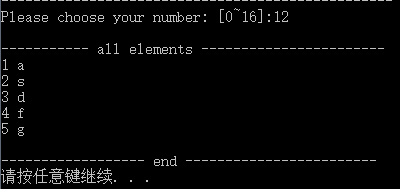
二十八、选择第三个表进行操作：（选择成功）



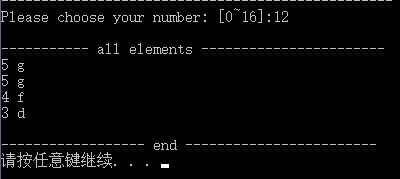
二十九、对第三个表进行初始化：输入测试数据：（初始化成功）



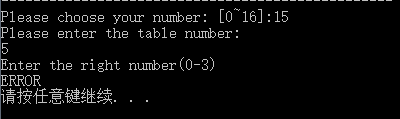
三十、重新返回第二张表并遍历：（表一遍历结果和切换表之前相同）



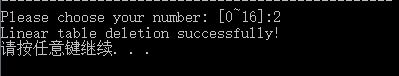
三十一、重新返回第一张表并遍历：（表一遍历结果和切换表之前相同，说明各表之间为独立操作）



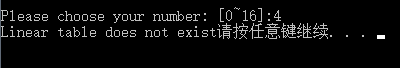
三十二、选择向第五张不存在的表转换：（操作失败，提示只能在1-3中进行选择，并报错）

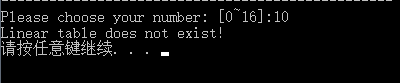


三十三、销毁第一张表：（销毁成功）



三十四、对第一张表进行各种操作（除初始化）：（所有操作都会不会进行并显示此表不存在。（因为操作太多，此处只列举二个方法的操作结果））





# 1.4 实验小结

通过这次实验，掌握了线性表的定义，顺序存储及基本操作。学会了利用线性表的顺序存储结构完成一些数据的管理，实现数据元素的增加，删除等运算。加深了对线性表的概念、基本运算的理解，掌握了线性表的基本预算的实现。熟练了线性表的逻辑结构和物理结构之间的关系。今后的学习过程中应当多从数据结构的角度分析如何进行数据的处理、存储以方便问题的解决。